TECHNICAL DOCUMENT 3148
December 2002

Infrared Search and Track Installation Instructions and User's Guide

DoD HPC Modernization Program (CHSSI SIP-8)

SSC San Diego
San Diego, CA 92152-5001

SSC SAN DIEGO
San Diego, California 92152-5001

Commanding Officer **Executive Director**

ADMINISTRATIVE INFORMATION

This report was sponsored by the Department of Defense High Performance Computing Modernization Office (HPCMO) Common HPC Software Support Initiative (CHSSI).

Released by

Active Acoustics

Under authority of

Maritime Surveillance Division

LH

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

The purpose of the Common High Performance Computing Software Support Initiative (CHSSI) Infrared Search and Track for Missile Surveillance (IRST) SIP-8 project is to provide a scalable, high performance capability for developing infrared surveillance algorithms. The basis for this effort is the Distributed Algorithm Stream (DAS) software package previously developed by the Airborne Infrared Measurement Systems (AIRMS) program, funded by the Defense Advanced Research Projects Agency (DARPA) Sensor Technology Office (STO) from 1989 through 1996. The AIRMS program demonstrated the longwave infrared (LWIR) sensor and signal processing technology necessary to detect dim targets at ranges of up to 300 nautical miles against cluttered backgrounds.

Under the AIRMS effort, DAS was originally programmed in the FORTRAN programming language and was run on Sun workstations, the VMS alpha and DEC workstations, and on the massively parallel Intel Paragon. The DAS processing software is a highly modular signal processing stream that includes data quality assessment, calibration, target injection, pattern noise mitigation, jitter mitigation, data rework, registration, 3D match filtering, and detection and tracking algorithms. The advantage conferred by the modular architecture of the IRST processing project is that individual algorithms may be modified without requiring changes to other algorithms.

The converted DAS software will be used as a testbed for improving various IRST algorithms. In order to evaluate the performance of the algorithms, the processing of a large amount of data is required. For instance, in order to determine that an algorithm has reached a false alarm rate of one false detection per hour, at least 10 hours of data must be processed. A scalable version of the algorithm could be run faster than real time on a parallel high performance computer in order to make such determinations in an acceptable amount of time. By making the software portable and scalable, development may be done on single workstations or small parallel machines, while large problems or large data sets may be processed efficiently on large high performance computing (HPC) platforms.

This document briefly describes the process of translating the FORTRAN alpha release DAS software modules to ANSI C using a commercial off-the-shelf (COTS) translator [Reference 1]. In addition to the DAS translation effort, the other critical task was selecting and implementing a communication library for the translated modules that would support wide portability and achieve a high degree of scalability on a number of different HPC platforms. Message Passing Interface (MPI) was selected as the parallelization tool for the translated DAS software modules, and a summary has been included that explains the selection of MPI. This document also provides a functional description of the converted DAS algorithms, instructions for installing the converted DAS software, instructions for compiling and linking the converted DAS software, and instructions for executing the test cases.

The original FORTRAN DAS software as developed by the AIRMS program is available from PAR Government Systems in La Jolla, CA (858) 551-9880.

2. DISTRIBUTION RESTRICTIONS

Distribution of the IRST software is limited to U.S. Government Agencies and their contractors. Furthermore, export of the IRST software is restricted by the Arms Export Control Act (Title 22, U.S.C., Sec 2751 et seq.). Violations of these laws are subject to severe criminal penalties.

3. CODE CONVERSION PROCESS

The DAS signal processing software package consists of approximately 500,000 lines of FORTRAN source code contained in more than 2,000 source files and 82 different directories. In addition, there are more than 130 DAS programs and 182 FORTRAN include files. The SIP-8 project is tasked with translating the DAS FORTRAN code to ANSI C and parallelizing the converted DAS software.

Because of the vast number of FORTRAN source lines and SSC San Diego's limited resources, a COTS translator was purchased from Cobalt Blue (FOR_C®) that was used to convert the DAS FORTRAN software into ANSI C. The Cobalt Blue translator produces readable and portable software. What distinguishes the Cobalt Blue translator from other COTS translators is that Cobalt Blue provides the source code for the translator's run-time library, thereby permitting porting of the translated software to different HPC platforms.

SSC San Diego developed a DAS conversion process document describing the process of translating the DAS FORTRAN software into ANSI C. DAS module SA_PN was the initial DAS module for demonstrating the conversion process and proved to be very successful. The DAS conversion process has subsequently been applied to all the converted DAS modules contained in the IRST distribution package. The success of the translation process was demonstrated not only by SSC San Diego but also by PAR Government Systems as well. The DAS conversion document is contained in the **irst_dist_xxx/doc** installation subdirectory.

The FOR_C run-time library must be built for each computer platform to which the DAS software is ported. To create the run-time library, simply compile all the C source files contained in the **irst_dist_xxx/code/cblue/fcrt** installation subdirectory, and the appropriate run-time library filename is created for the selected platform. The run-time library filename is specified as a dependency of the translated software. The ability to re-compile the FOR_C run-time library for each HPC platform thereby provides portability. For specific instructions on how to build the FOR_C run-time library, refer to Section 6.5, "Compiling and Linking."

In order for the translator to function properly and produce high-quality translation results, the original FORTRAN software must be structured FORTRAN. In addition to the FOR_C translation software tool, Cobalt Blue also developed a FORTRAN Structuring tool (FOR_STRUCT®), which is a comprehensive structuring utility that transforms spaghetti software into fully structured software. When executing program FOR_STRUCT on the DAS FORTRAN, a more conventional ANSI C code is produced.

This document does not address the FOR_C translation results because there are simply far too many subtle translation artifacts to describe. However, it should be noted that the I/O translation results initially appear to be obscure because calls are made to the FOR_C run-time library. In reviewing these calls more closely, it is clear that the calls are nothing more than Cobalt Blue C functions that perform I/O [Reference 1].

After the DAS code has been successfully translated, the ANSI C DAS code is checked into Concurrent Version System (CVS). The translated DAS code is then compiled and compilation errors are resolved. Typical translation errors are as follows: non standard-intrinsic functions or other non-standard language features, or where calls to the underlying operating system are not portable.

After the code has been compiled successfully, a commercial lint tool called Flexelint from Gimpel Software was executed on the converted code. Flexelint has proven to be a valuable asset in

finding redundant code and redundant variable declarations, and when local code changes cause non-local problems. After all the Flexelint errors and warning messages have been resolved, the converted module is executed and the ANSI C DAS created output file is validated.

4. PARALLELIZATION PROCESS

One issue that the CHSSI signal and image processing projects must address is the possibility that the software could be used in an embedded system on, for instance, a ship or aircraft, to process sensor data in real time. It has been shown that the DAS software would require parallelization in order to process a typical IR sensor in real time. In addition, even in a non-real-time application such as an algorithm testbed, high-speed processing is needed in order to process larger data sets for algorithm performance analysis. Without the capability to process large amounts of sensor data, algorithm developers may be limited to showing good performance for only a selected set of cases. A parallel implementation of the test bed may allow enough data to be processed to determine the true performance of the algorithm.

To implement the parallelism of the converted IRST software, two interface alternatives were considered: MPI and Scalable Programming Environment (SPE). For communications portability among parallel HPCs, the MPI Standard library has become widely accepted. The vendors of nearly all HPC systems of interest provide high performance implementations of MPI. MPI is also available on the major embedded systems platforms. Alternatively, SSC San Diego has developed the SPE, a high-level communication library that has been ported to a number of HPC platforms [Reference 3]. In addition to point-to-point messaging, the SPE will transparently distribute data to parallel processors. The SPE has been used successfully in a number of signal processing projects.

During the AIRMS project, the most important modules of the DAS signal processing software were modified to run in parallel on the Intel Paragon and on a network of workstations. This was primarily done through an image tiling interface, although various direct communication calls occur throughout the code. This parallel functionality was implemented with direct calls to both the proprietary Intel communications library (NX) and to the Parallel Virtual Machine (PVM) library. With the demise of the Intel Paragon, the NX library has become obsolete. PVM is an open source library that provides portable communications among workstations. Although widely portable, PVM is unsuitable for high performance message passing within the processors of a parallel machine due to the relative inefficiency of communications compared to the manufacturers optimum method on that machine.

The approach to parallelizing the DAS software, then, was a choice between directly modifying the NX and PVM library calls into MPI library calls, and more substantially modifying the DAS code to use the SPE library. Given the size and complexity of the DAS software, it was decided to use the former approach.

5. DAS MODULES IN BETA DELIVERY

5.1 OVERVIEW OF DAS BASELINE PATHWAY

The three basic processing steps for DAS are (1) pre-processing, (2) registration and filtering, and (3) detection and tracking. While alternate modules exist in DAS for most of the steps in the processing stream, the DAS Baseline modules represent the preferred pathway through these three steps because these modules are the most useful, thoroughly tested, and fully validated modules [Reference 2]. Figure 1 shows the modules and functionality of the Baseline pathway. The data input at the beginning of the pathway consists of raw AIRMS-collected imagery or synthetically generated data. Those modules indicated in red and marked accordingly are the modules present in this delivery, namely:

- Target Insertion (AP_TI)
- Sensor Artifact Removal (SA_PN)
- Registration (RS_RS)
- Covariance Based Filter (FL_IR)
- Reverse Registration (RS_RS_ST)
- Velocity Stacking (FL_SP_VD)
- Detection/Thresholding (DT_BN)

5.2 OVERVIEW OF DELIVERED DAS MODULES

The following sections contain brief functional descriptions of the DAS modules that were converted and parallelized by SSC San Diego and PAR Government Systems, and represent the DAS modules contained in the IRST Beta software distribution package. Operation of the DAS modules is controlled via NAMELIST files. The NAMELIST files contain input parameters necessary to execute the DAS modules.

For guidelines on individual DAS module parameter settings and a description of the DAS algorithm, the user is directed to the individual module description documents contained in the **irst_dist_xxx/doc/irst_doc** installation subdirectory. In addition to the individual DAS module description documents, subdirectory **irst_dist_xxx/doc/irst_doc/Final_Report** contains AIRMS Final Report [Reference 4].

5.2.1 Module AP_TI

AP_TI inserts one or more additive point targets into the scene. The user specifies via the namelist the intensity (watts per steradian), position (pixels in azimuth and elevation), velocity (pixels per frame in azimuth and elevation) and range (kilometers) of each target. Additionally, the user must specify sensor characteristics, such as the point spread function (PSF), instantaneous field-of-view (iFOV) and the spectral bandwidth. For a more detailed description of the AP_TI algorithm, the user is directed to the AP_TI module description document and the AIRMS Final Report.

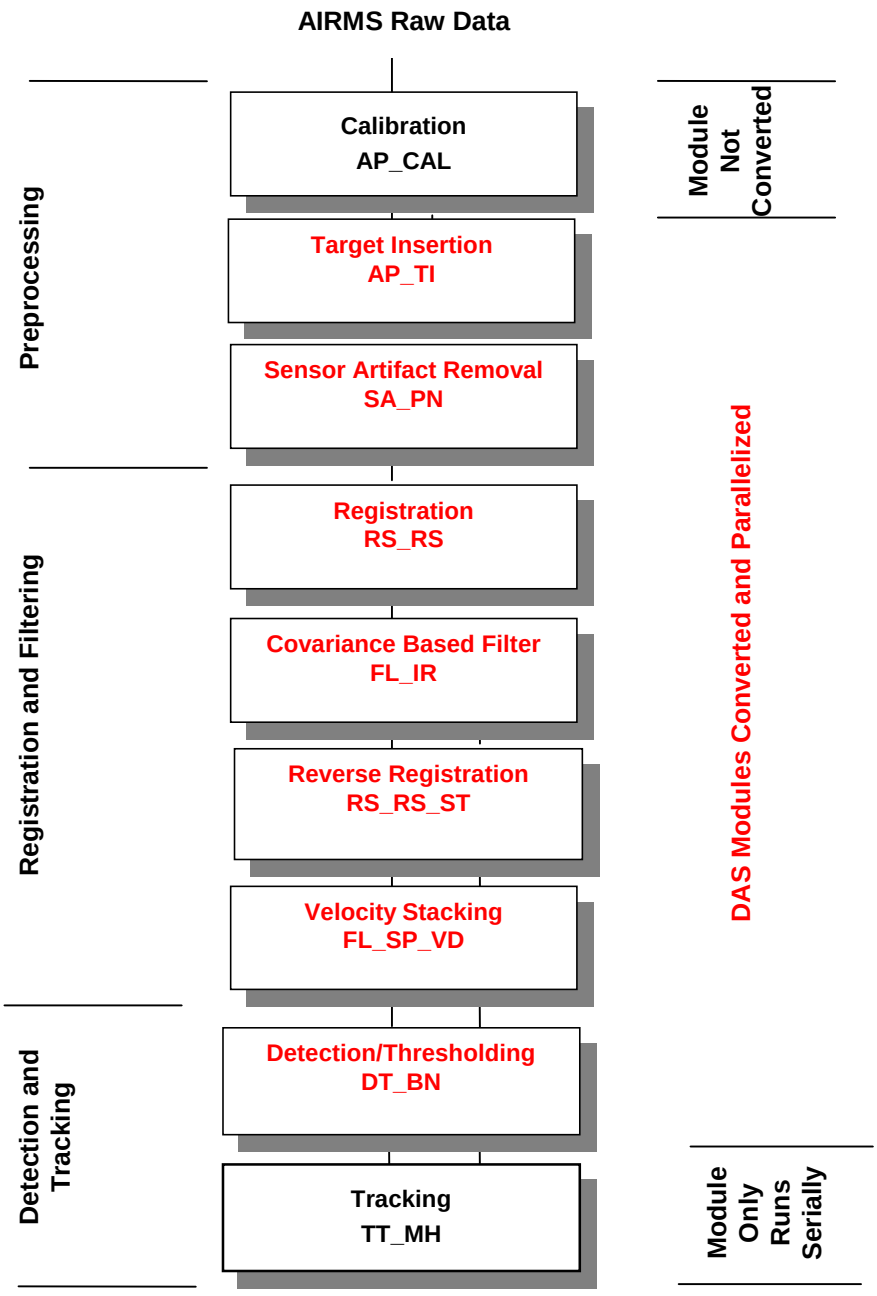

Figure 1. DAS Baseline pathway.

5.2.2 Module SA_PN

SA_PN mitigates the pattern noise associated with the AIRMS IRST sensor. The pattern noise consists of three main components.

The dominant noise component is very typical of all scanned electro-optic sensors and consists of a random bias from row-to-row that slowly varies across the image. This component is efficiently

suppressed by notching energy whose spatial frequency is in a narrow band along the azimuthal DC axis. This notch does not, however, extend to k_{el}=0 to avoid significantly distorting the clutter content. This is implemented by (1) low-pass filtering in azimuth to estimate the local bias for each row, (2) high-pass filtering in elevation to retain only the variation from row-to-row, and (3) subtracting the resulting pattern estimate from the observed image. The namelist parameter "DO_BIAS" exercises this option.

A similar philosophy is used for the two other fixed pattern components that are peculiar to the AIRMS sensor. An azimuthally white bias which repeats every 5-lines (namelist parameter "DO_FIVE_LINE") and a series of "steps" that occur every 25-lines, corresponding to multiplexor (MUX) groupings in the readout electronics (namelist parameter "DO_MUX_BIAS"). For a more detailed description of the SA_PN algorithm, the user is directed to the SA_PN module description document and the AIRMS Final Report.

5.2.3 Module RS_RS

The intent of the RS_RS module is to align similar clutter features in a stack of frames. This spatial consistency over time results in better temporal cancellation in the matched filter that follows (for which FL_IR is the baseline module). RS_RS is described as a "rubber sheeting" image registration. Conceptually, it can be viewed as stretching one image as though it were a rubber sheet so that it comes into alignment with the reference image. This is a two-step process. First, the corresponding points in the two frames are determined by maximizing a *match metric* that is based on the likelihood of the observations given a hypothesized displacement. Second, the geometric mapping described by the resulting *match points* is used to resample the image to accomplish the actual warping. The combination of these two steps must produce a set of match points that are consistent and are sensitive to the overall content of the image. The desire to achieve a significant degree of temporal cancellation drives both the required accuracy for the match points and the fidelity of the interpolation employed in resampling the intensities.

The DAS rubber-sheet registration algorithm has two key features. First, it uses a hierarchical process that generates frame-to-frame correspondences on a coarse basis, using decimated low-spatial-resolution imagery. The information from this coarse registration then guides the algorithm at finer resolutions, leading to the ultimate full resolution match points. Advantages of this approach include computational efficiency and a framework in which low clutter-free regions of an image (e.g., blue sky) are not distorted by registration. Second, it uses a simple clutter and noise model, which represents key features of the data, without producing a computationally complex implementation. This model drives a maximum-likelihood algorithm leading to a robust match point selection.

The RS_RS module contains many software switches that try to enhance and elaborate on the basic algorithm steps of match point grid generation, match point interpolation, and resampling.

The DAS registration algorithms are the most difficult and provide the most functionality of all the DAS modules. For a more detailed description of the registration algorithm, the user is directed to the RS_RS module description document and the AIRMS Final Report.

5.2.4 Module FL_IR

FL_IR designs and applies the velocity independent portion of a continuously adaptive three-dimensional matched filter using the Point Spread Function (PSF) and the estimate of the clutter

covariance matrix. The algorithm begins by locally demeaning the data with a 3-D moving “cube” that is typically 5x7x3 pixels. Next, the sample covariance matrix is estimated for each homogeneous region in the scene. The filter can be designed and applied using the data from the whole image (no segment map is input in the namelist) or, alternately, separate filters can be designed for regions defined via the input segment map. The segment map is designed using the DAS module RS_SG that is currently scheduled for implementation next year. The resulting covariance matrix (or matrices for a multi-segment case) is computed by averaging the covariance matrices of cubes of data belonging to the region of interest. The algorithm then inverts the matrices and multiplies the results by cubes of image data in order to spectrally whiten the scene. The whitened data is then convolved with the PSF.

As with RS_RS, FL_IR is a complex algorithm. For a more detailed description of the covariance filtering algorithm, the user is directed to the FL_IR module description document and the AIRMS Final Report.

5.2.5 Module RS_RS_ST

The intent of the RS_RS_ST module is to move potential targets in the scene back to their original “nonjittered” positions so that the targets will be properly integrated during frame stacking (FL_SP_VD module). To accomplish this purpose, RS_RS_ST can either restore the nonjittered targets to their exact positions prior to RS_RS or it can restore jittered targets to positions based on the pixel velocities provided by RS_RS. For a more detailed description of the reverse registration algorithm, the user is directed to the RS_RS module description document and the AIRMS Final Report.

5.2.6 Module FL_SP_VD

The heart of the DAS signal processing stream is the 3-D matched filter modules used to suppress clutter and enhance target responses. Both the FL_SP and FL_SP_VD modules apply 3-D matched filtering for target detection. These filtering operations are divided between the two modules with FL_SP applying velocity-independent operations of 3-D matched filtering while FL_SP_VD applies the velocity dependent operations by integrating filtered target energy over a number of shifted frames according to a set of target velocity hypotheses.

The clutter motion is restored in the Reverse Registration module RS_RS_ST and the restored data is passed to FL_SP_VD where target energy is summed or integrated under a number of velocity hypotheses. Because this integration of target energy is performed by shifting and interpolating the image frames along a grid of velocity hypotheses, velocity-dependent operations are known as “stacking”, hence the FL_SP_VD module is also know as the “stacker.” For a more detailed description of the FL_SP_VD algorithm, the user is directed to the FL_SP_VD module description document and the AIRMS Final Report.

5.2.7 Module DT_BN

The purpose of the Detection and Thresholding, Background Normalization (DT_BN) module is to provide the subsequent tracker module with constant false alarm rate (CFAR) detection information that will aid its function in forming target tracks and minimizing false tracks (e.g. clutter related tracks). DT_BN also provides detection information that can be used to generate ROC curves that measure the performance of the software modules that precede DT_BN in DAS. For a more

detailed description of the DT_BN algorithm, the user is directed to the DT_BN module description document and the AIRMS Final Report.

5.2.8 Module PARALLEL DAS

All of the converted DAS modules are capable of being executed either sequentially or in parallel mode when using MPI. The flexibility of running the DAS modules either sequentially or in parallel has been preserved from the original DAS FORTRAN. Parallel DAS is executed by calling module *das*. For sequential module execution of the DAS modules, execute the specific module binary, e.g., *prog_ap_ti*. Unlike serial DAS that requires outputs to be written to files between one DAS module to the next DAS module in the pipeline, parallel DAS has the advantage of processing multiple stacks of data from a file with a single call to the *das* executable.

5.3 DAS ANALYSIS MODULES

In addition to the already mentioned converted and parallelized DAS modules, two DAS analysis modules have also been converted to ANSI C from FORTRAN – *bas2eos* and *mat2bas*. These two modules can assist the researcher to analyze the output of the DAS algorithms.

5.3.1 Module BAS2EOS

Module *bas2eos* converts a BASIL I/O file to Electro-Optical Systems (EOS) format. The EOS file is defined by its width and height, the size of any margins surrounding the image (specified by the margins containing only valid pixels of the total image), a value specifying in which of several formats the actual image data is stored, and the image data. The EOS file also establishes the structure for filters, which are identical to images except smaller in size. This module is executed when performing data verification on the BASIL I/O files and is not normally invoked directly by a user.

5.3.2 Module MAT2BAS

Module *mat2bas* converts a MATLAB file to BASIL I/O format. Module *mat2bas* has an option of converting all frames or a selected index of the frames contained in the MATLAB file into BASIL format. This module allows users to provide their own data for testing.

6. DISTRIBUTED ALGORITHM STREAM OPERATION

6.1 PLATFORMS

The converted and parallelized IRST signal processing modules and supporting files have been installed and fully tested on the systems defined in Table 1.

Table 1. HPC platforms.

Hewlett-Packard Superdome

Location:	SSC San Diego Distributed Center
Name:	Longview
Operating System:	HPUX 11i
Number of Nodes:	48
Speed of Processors:	550 MHz

SUN Ultra Enterprise 1000

Location:	Naval Oceanographic Office (NAVO)
Name:	Wolfe
Operating System:	SunOS 5.7
Number of Nodes:	64
Speed of Processors:	400 MHz

Silicon Graphics Origin®2000

Location:	Air Force Aeronautical Systems Center (ASC)
Name:	hpc03-1
Operating System:	IRIX 6.5
Number of Nodes:	23
Speed of Processors:	195 MHz

Silicon Graphics Origin 2000

Location:	SSC San Diego
Name:	Puma
Operating System:	IRIX 6.5
Number of Nodes:	1
Speed of Processors:	195 MHz

Mercury

Location:	AFRL/SN
Name:	Mercury MP-510
Operating System:	MC/OS 5.7.0
Number of Nodes:	64
Speed of Processors:	400MHz

Linux Cluster

Location	Air Force Research Lab/IF
Name	Hades
Operating System	Linux
Number of Nodes	32-48
Speed of Processors	700MHZ and 1300MHZ Athalons (Pentium II)

6.2 DISTRIBUTION PACKAGE CONTENTS

Table 2 lists the files included in the IRST distribution package, where xxx is the release number.

Table 2. Distribution package contents.

irst_dist_xxx	Directory for IRST installation
EXPORT_CONTROLLED	Export disclosure notice
README	ASCII IRST Introduction
irst_dist_xxx/code	Directory for IRST files
Makefile	Compiles the IRST distribution package
irst_dist_xxx/code/src	Directory for IRST source files
irst_dist_xxx /code/cblue	Directory for Cobalt Blue
README	Cobalt Blue compilation instructions
include file (*.h)	Include files
html files (*.html)	HTML files
irst_dist_xxx /code/cblue/fcrt	Directory for Cobalt Blue source code
mkfcrt	Builds translation run-time library
source files (*.c)	Translator source files
irst_dist_xxx /code/env	Directory for platform compiler options
hp	Hewlett Packard
irix	Silicon Graphics
linux	Linux
mercury	Mercury
solaris	Sun
solaris.NAVO-wolfe	NAVO Sun
solaris.nephi.mpich	SSC SD-Nephi
irst_dist_xxx /code/include	Directory for IRST include and prototype files
include files (*.h)	IRST include files
INCLUDE_LIST	List of all include files
irst_dist_xxx /code/lint	Directory contains platform specific Flexelint files (*.lnt)
irst_dist_xxx /doc	Directory for documents
incident_log	IRST details of software fixes
users_manual.pdf	IRST Users Manual
conversion.pdf	IRST FORTRAN to ANSI C
Irst_dist_xxx/doc/irst_doc	Directory for IRST algorithm documents (*.pdf)
irst_dist_xxx/doc/irst_doc/Final_Report	Directory for AIRMS final report (*.pdf)
Irst_dist_xxx/test	Directory for IRST validation
comp_stats_irst.c	Comparison statistics
go	Executes test cases using different options
runtests	Executes all test cases for a given directory
Makefile	Compiles file comp_stats_irst.c
irst_dist_xxx/test/AP_TI/	Directory for AP_TI test cases
Irst_dist_xxx /test/DAS/	Directory for pipeline test cases
irst_dist_xxx/test/DT_BN/	Directory for DT_BN test cases
irst_dist_xxx/test/FL_IR/	Directory for FL_IR test cases
irst_dist_xxx/test/FL_SP_VD/	Directory for FL_SP_VD test cases
irst_dist_xxx/test/MAT2BAS/	Directory for MATLAB to BASIL test case
irst_dist_xxx/test/RS_RS/	Directory for RS_RS test cases
irst_dist_xxx/test/RS_RS_ST/	Directory for RS_RS_ST test cases
irst_dist_xxx/test/SA_PN/	Directory for SA_PN test cases
irst_dist_xxx /data/AP_TI/	Directory for AP_TI reference files
irst _dist_xxx /data/Cloud_data/	Directory for input Cloud data files
irst _dist_xxx /data/DT_BN/	Directory for DT_BN reference files
irst _dist_xxx /data/FL_IR/	Directory for FL_IR reference files
irst _dist_xxx /data/FL_SP_VD/	Directory for FL_SP_VD reference files
irst _dist_xxx /data/MAT2BAS	Directory for MAT2BAS reference files
irst_dist_xxx /data/Parameters/	Directory containing input auxiliary files
irst _dist_xxx /data/RS_RS/	Directory for RS_RS reference files
irst_dist_xxx /data/RS_RS_ST/	Directory for RS_RS_ST reference files
irst_dist_xxx /data/SA_PN/	Directory for SA_PN reference files

6.3 INSTALLATION AND CONFIGURATION

Installation of the IRST distribution package requires at least 50 MB of disk space. Additional disk space will be required to store the processed output BASIL I/O and log files. These instructions

assume that you have an intermediate knowledge of UNIX command syntax and experience installing applications on UNIX. If you have any problems, please contact your system administrator for technical assistance.

Copy the IRST distribution package (irst_dist_xxx.tar.Z) to the directory for installation and perform the following steps:

uncompress irst_dist_xxx.tar.Z
tar xvf irst_dist_xxx.tar

The preceding two steps will install all of the IRST software and supporting documentation in the **irst_dist_xxx** subdirectory of the current installation directory (referred to as the **install_dir**).

Create the required environment variables. The environment variable “IRST_ENV” defines the Unix machine for operation, “IRST_DIR” defines the location of the IRST software, and “IRST_TEST_DATA_DIR defines the data location.

setenv IRST_ENV [hp, irix, linux, mercury, solaris, solaris.NAVO-wolfe, or solaris.nephi.mpich]
setenv IRST_DIR **install_dir/irst_dist_xxx**
setenv IRST_TEST_DATA_DIR **install_dir/irst_dist_xxx/data**

Create a temporary subdirectory for writing the DAS output BASIL data files as follows:

mkdir **install_dir/irst_dist_xxx/temp**

Upon execution of a DAS module, a unique subdirectory name is created in directory **install_dir/irst_dist_xxx/temp**, which will contain the binary BASIL output file and other ASCII log files.

If there is insufficient file space on the default disk to contain the output BASIL files, then modify variable TEMP_DIR in file **install_dir/irst_dist_xxx/code/$IRST_ENV** to redirect the output BASIL files.

The Makefiles contained in the IRST source distribution directories may update more than one target at a time when supported by the host computer. For instance, on the HP Superdome the number of targets updated concurrently is determined by the environment variable PARALLEL. Typically, PARALLEL is set to the maximum number of available processors.

setenv PARALLEL n

The user may find it convenient to create a shell script that defines the aforementioned environment variables so as to avoid the need to create them individually after each login.

6.4 CONFIGURATION

The DAS software has been designed to allow researchers to compile DAS using several different configurations, depending on supporting software installed on the target platform and the researcher’s goal. The DAS software is highly configurable and supports MPI and the Vector, Signal, and Image Processing Library (VSIPL). The following configurations are available to the researcher:

- MPI and VSIPL
- MPI and No VSIPL
- No MPI and VSIPL
- No MPI and No VSIPL

The DAS program *mat2bas* requires the installation of Matlab libraries. Because all HPC platforms may not contain the required library files for compiling program *mat2bas*, program *mat2bas* is also configurable. To execute all the DAS programs included in the IRST distribution, both MPI and Matlab must be installed on the target platform.

6.4.1 MPI

Installation of MPI is required to create and execute the parallel DAS programs. Configuring the DAS software to include MPI is performed by editing the environment file **install_dir/code/env/$IRST_ENV** according to the comments in the file. If MPI is to be included in the compilation process, then the appropriate path names for MPI must be included in the **install_dir/code/env/$IRST_ENV** file. If MPI is not installed on the target platform or there is no interest in executing the parallel DAS programs, then simply comment out the lines that reference MPI as instructed in the **install_dir/code/env/$IRST_ENV** file. By excluding MPI from the compilation process, only the sequential DAS programs are capable of being executed.

6.4.2 VSIPL

In addition to supporting MPI, DAS also support VSIPL. Configuration of VSIPL is also performed by editing the environment file **install_dir/code/env/$IRST_ENV** according to the comments in the file. If VSIPL is to be included in the compilation process, then the appropriate path names for VSIPL must be included in the **install_dir/code/env/$IRST_ENV** file.

6.4.3 Matlab

Program *mat2bas* requires the existence of Matlab library files for successful compilation. If the Matlab library files do not exist on the target platform, then simply edit the **install_dir/code/env/$IRST_ENV** and comment out lines that reference Matlab. Also follow the instructions in the **install_dir/code/env/$IRST_ENV** file.

6.5 COMPILING AND LINKING

At a minimum, compilation of the IRST distribution package requires the presence of the following pre-installed software:

C compiler specific to the platform

Bourne shell or bash shell (Linux only)

Please note that the compilation anticipates standard location for compilers, libraries, and system utilities, such as *make*. However, to fully exploit the parallel implementation of DAS, it is recommended that MPI be installed. In addition, program mat2bas requires the presence of Matlab library files for the compilation process. Please refer to Section 6.4 “Configuration”, for specific instructions on configuring the DAS software. All system specific locations are changed in the **install_dir/code/env/$IRST_ENV file.**

Create the following environment variables:

setenv cblue **install_dir/irst_dist_xxx/code/cblue**
setenv os [hpux, iris, sparc, linux, or mercury]

The environment variable “cblue” defines the location of the Cobalt Blue software, and “os” defines the Unix machine for the Cobalt Blue run-time library.

Three separate steps are required for compiling and linking the IRST software. The first step is to build the Cobalt Blue library, which is performed by executing the following commands:

1. **cd install_dir/irst_dist_xxx/code/cblue/fcrt**
   sh *mkfcrt*

The above command compiles and links all of the Cobalt Blue software for the defined operating system contained in the environment variable “os”. Depending on the operating system, the output Cobalt Blue library filename is fcrtx.a, where x = i for iris, or x=s for sparc, x=h for hpux, x=lx for linux, or x=mc for mercury.

After successfully building the Cobalt Blue library, execute the following commands:

2. **cd install_dir/irst_dist_xxx/code**
   make

The above commands compile and link all of the IRST distribution software. As a result of successfully executing this command, all of the IRST image processing executables will reside in subdirectory **install_dir/irst_dist_xxx/code/src/prog_yyy**, where yyy identifies the DAS module.

The third and final step is to compile the verification and validation module *comp_stats_irst*. Execute the following commands:

3. **cd install_dir/irst_dist_xxx/test**
   make

If individual IRST modules require re-compilation, then simply execute the Makefile contained in the IRST program directories.

6.6 DATA DESCRIPTION

The AIRMS program successfully collected over 1.5 terabytes of image data, which was processed through DAS using the Paragon HPC implementation. For internal software validation conducted at SSC San Diego, two AIRMS data sets were processed through the converted and parallelized DAS modules, Cloud Data and Terrain Data. This distribution package only contains the Cloud Data set from the AIRMS database, which was used for testing and verifying the converted and parallelized IRST modules included in the distribution package. Table 3 provides a brief data description of the Cloud Data.

The I/O format of the DAS image files is named Baseline Algorithm Stream Input/Output Logic (BASIL) [Reference 5]. BASIL is the standard I/O package used by the DAS image processing software. All of the converted DAS routines are capable of reading and writing BASIL files. BASIL was developed to provide a consistent, flexible and extendible I/O architecture that could grow along with the needs of the AIRMS system and the data processing requirements.

Table 3. Data description.

Name Prefix	Number of Frames	Pixels in Azimuth	Pixels in Elevation	Clutter Type	Real Targets/ Type	Comments
'cd'	5	256	256	Cloud/sky	None	A subpatch from Flight 12 (12:46:11), clouds at 20 – 30 km altitude. A/C altitude 39,000 feet.

6.6.1 Input Data

The FORTRAN-generated BASIL input files (provided in the distribution package) are used for executing the converted and parallelized DAS modules. Because each module is capable of running sequentially, the distribution package contains at least one BASIL input file for each DAS translated module. In addition, there are also FORTRAN generated auxiliary files contained in the distribution package, which are required by some of the translated DAS modules.

6.6.2 Reference Data

In order to verify that the newly installed, translated, and parallelized software creates BASIL output files consistent with the FORTRAN-generated BASIL files, the distribution package contains reference files that were generated using the DAS FORTRAN. The FORTRAN generated reference files are used to verify the validity of the BASIL output files created by the translated modules. A standard verification test program is included that validates each newly created BASIL output file by quantifying the pixel differences between the two BASIL files.

7. DAS MODULE TESTING

Execution of the converted and parallelized DAS modules is performed through the execution of program *go* or *runtests*, which require at least one input argument, the test case filename. A number of different test cases have been designed and created for the execution of the converted DAS modules. The test cases are contained in the test subdirectories that are identified by their DAS module name. Because DAS was originally programmed in FORTRAN, the DAS modules received their inputs from namelist files, which are also referred to as test cases within the confines of this document. The test cases were created to test different software paths for the converted and parallelized DAS modules, and to demonstrate that the number of processors for DAS execution is a configurable parameter. Each test case filename defines a particular test for the executed DAS module.

In addition to running a specific DAS module either sequentially or in parallel mode, the DAS modules can also be run in a pipeline mode. For pipeline execution of the DAS modules, the test cases are contained in the DAS subdirectory.

Because the Terrain Data is not contained in the IRST software distribution package, none of the test cases referencing the Terrain Data set can be executed successfully. In addition, only the FORTRAN reference files for the Cloud Data test cases ending with a 1 (e.g. cd_seq_1) are contained in the IRST distribution package, and as a result only those Cloud Data test cases can be executed successfully. Use caution when executing *runtests* for an entire subdirectory to avoid

executing test cases for which there is no input or reference data – all of the Terrain Data test cases, and Cloud Data test cases ending in a number other than 1.

7.1 DAS DRIVER PROGRAMS

Contained in the **install_dir/irst_dist_xxx/test** directory are programs *go* and *runtests*, which are used to execute the test cases for the converted and parallelized DAS modules. Programs *go* and *runtests* expedite the execution of the converted DAS modules by processing the DAS configurable parameters contained in the test cases.

7.1.1 *go*

The *go* program executes a single DAS test case test file, which is specified on the program execution command line. Execution of program *go* with no options displays the following usage note:

```
Usage: go [options] testspec
  Arguments:
    -q        quiet -- output to file rather than screen
    -t        show timing information
    -tv       run the program under TotalView
    -x        run the program but do not validate the result
    testspec  testspec file to be used to drive execution
```

The only required argument for the *go* program is testspec, which identifies the test case filename. When executing the *go* program, a subdirectory is created in the temp directory, which contains all the DAS output files. The naming convention for the created subdirectory is *pp_seq_nnnn*, where "*pp*" is the program name, "*seq*" identifies sequential execution, and "*nnnn*" is the process number. The naming convention for the parallel subdirectory is *pp_par_nnnn*, where "*par*" identifies parallel execution.

The default execution of the *go* program always performs verification of the ANSI C created output BASIL file, using program *comp_stats_irst*. Program *comp_stats_irst* compares the pixel differences between the newly created ANSI C BASIL output file to the FORTRAN-generated BASIL reference file. The difference image is normalized, and program *comp_stats_irst* produces a report giving the maximum pixel difference, the standard deviation, and the mean of the normalized difference values.For example, execution of SA_PN using test case cd_seq_1 is performed as follows:

cd install_dir/irst_dist_xxx/test
***go* SA_PN/cd_seq_1**

Figure 2 shows actual output of an SA_PN execution on the Hewlett Packard Superdome, located at SSC San Diego. SA_PN starting in sequential mode.

```
SA_PN_CHILD: Total processing time = 2.599 seconds
SA_PN execution time: 2.893 seconds.
SA_PN completed successfully.
BAS2EOS: dumping BAS file.
Image size: 256 rows by 256 cols
BAS2EOS: complete.

comp_stats_irst difference statistics: PASSES
comparison to requested values less than 1.0e-02 (relative)
maximum:  9.578471e-07
mean:    -9.682219e-08
stddev:   2.008243e-07
comp_stats_irst difference statistics: PASSES
comparison to requested values less than 1.0e-02 (relative)
maximum:  9.614707e-07
mean:    -9.811202e-08
stddev:   2.017612e-07
comp_stats_irst difference statistics: PASSES
comparison to requested values less than 1.0e-02 (relative)
maximum:  9.539805e-07
mean:    -9.711206e-08
stddev:   1.997765e-07
comp_stats_irst difference statistics: PASSES
comparison to requested values less than 1.0e-02 (relative)
maximum:  1.157750e-06
mean:    -9.834672e-08
stddev:   2.040039e-07
comp_stats_irst difference statistics: PASSES
comparison to requested values less than 1.0e-02 (relative)
maximum:  1.152715e-06
mean:    -9.886634e-08
stddev:   2.018421e-07
```

Figure 2. SA_PN output example.

7.1.2 *runtests*

In addition to program *go*, program *runtests* is capable of executing a given test case or all the test cases found in the subdirectories. Execution of program *runtests* with no options displays the following usage note*:*

Usage: runtests [-t] arguments [...]
Executes the IRST test on each argument or file found in subdirectories.
-t Show timing information, not pass/fail.

Program *runtests* was created to expedite the execution of several test cases without providing each of the test case filenames for each DAS module. By providing a wildcard "*" with the test case filename a number of different test cases are executed. For example, to execute all the test cases contained in all the subdirectories, simply type *runtests* "*". The image fidelity test for each test case will produce a "pass" or "fail" message to the display without giving the overall statistics.

For example, execution of program *runtests* for SA_PN using all Cloud Data test scripts ending in 1 is performed as follows:

cd install_dir/irst_dist_xxx/test
***runtests* SA_PN/cd*1**

The following is an actual output of program *runtests* on the Hewlett Packard Superdome, located at SSC San Diego:

```
Test "SA_PN/cd_03node_1" ......................... passed
Test "SA_PN/cd_18node_1" ......................... passed
Test "SA_PN/cd_seq_1" ............................ passed
```

7.2 SINGLE MODULE AND PIPELINE DAS TEST CASES

The test cases for executing a single DAS module are contained in the test subdirectories, which are named for the DAS module (e.g., **install_dir/irst_dist_xxx/test/SA_PN)**. The file naming convention for the single-module test cases is *dd_seq_n*, where "*dd*" is either cd for Cloud Data or td for Terrain Data, "*seq*" is sequential, and "*n*" is the unique test case number. Naming convention for the parallel test case filenames is *dd_ppnode_n*, where "*pp*" is the number of processors allocated.

In addition to running a single DAS module, pipeline test cases for the parallel DAS modules exist in directory **install_dir/irst_dist_xxx/test/DAS**. Parallel DAS has the advantage of processing multiple stacks of data from a file with a single call to the *das* executable, unlike the sequential pipeline that requires file outputs to be passed from one DAS module to the next. The file naming convention for the test cases contained in the DAS subdirectory is *dd_first_last_ppnode_n*, where "*dd*" is cd for Cloud Data or td for Terrain Data, "*first*" is the first module in the pipeline, "*last*" is the last module in the pipeline, "*pp*" is the number of processors allocated, and "*n*" indicates a unique test case number. For details of the DAS data flow for the converted and parallelized modules, refer to Figure 1.

The pipeline test cases are also executed using program *go* or *runtests.* All of the Cloud data test cases can be run successfully from the IRST distribution DAS subdirectory.

8. DAS SOFTWARE LIMITATIONS

This section briefly describes FORTRAN DAS configurations that are not supported by the translated and parallelized DAS programs.

1. Program *das*, for FL_IR "checkerboard" option, will halt and print and error message when the following NAMELIST parameters are set:
   NODES_AZ > 1,
   NODES_EL > 1, and
   CVM_GEN_OPT = 3

2. Program serial FL_SP_VD will halt and print an error message when PROJ_FLAG=TRUE. The original DAS FORTRAN sets this variable to FALSE when Fl_SP_VD is executed in parallel mode.
3. In order to compile program *mat2bas*, the following MATLAB library files must be installed on the targeted platform for DAS execution.

    libmat.sl
    libmi.sl
    libmx.sl
    libut.sl

If these MATLAB library files do not exist on the targeted platform for DAS execution, then program *mat2bas* will not compile. This is a stand-alone program that is used to convert MATLAB files into BASIL files, and therefore will not affect execution of the DAS modules.

9. REFERENCES

1. Cobalt Blue Inc., “FORTRAN to C Translator,” February 1999.
2. Dennis Cottel, “Staged Module Development Plan for the CHSSI SIP-8 Infrared Search and Track Project,” Internal Project Memo, SSC San Diego, August 2000.
3. Partow, P., and D. Cottel. “The Scalable Programming Environment.” SSC San Diego Technical Report 1672, Rev. 1, September 1995.
4. PAR Government Systems Corporation, “Airborne Infrared Measurement System (AIRMS) Final Technical Report,” December 1996.
5. PAR Government Systems Corporation, “BASIL User’s Manual,” December 1996.

INITIAL DISTRIBUTION

Defense Technical Information Center
Fort Belvoir, VA 22060–6218 (4)

SSC San Diego Liaison Office
C/O PEO-SCS
Arlington, VA 22202–4804

Center for Naval Analyses
Alexandria, VA 22311–1850

Office of Naval Research
ATTN: NARDIC (Code 362)
Arlington, VA 22217–5660

Government-Industry Data Exchange
Program Operations Center
Corona, CA 91718–8000